PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F 17/30 — A1

(11) International Publication Number: WO 96/29663

(43) International Publication Date: 26 September 1996 (26.09.96)

(21) International Application Number: PCT/US96/03650

(22) International Filing Date: 18 March 1996 (18.03.96)

(30) Priority Data:
08/406,360 17 March 1995 (17.03.95) US
08/566,281 1 December 1995 (01.12.95) US

(71) Applicant: MICROSOFT CORPORATION [US/US]; One Microsoft Way, Redmont, WA 98054 (US).

(72) Inventors: BLUMER, Thomas, P.; 63 Sacramento Street, Cambridge, MA 02138 (US). AMSTEIN, Peter, R.; 1115 Castro Street, San Francisco, CA 94114-2513 (US). DRELLISHAK, Scott, F.; 1231 Vincente Drive #52, Sunnyvale, CA 94086 (US). FORGAARD, Randy, J.; 22 Fottler Avenue, Lexington, MA 02173 (US). SCHULERT, Andrew, J.; 267 Allston Street, Cambridge, MA 02139 (US).

(74) Agent: GORDON, Peter, J.; Wolf, Greenfield & Sacks, P.C., 600 Atlantic Avenue, Boston, MA 02210 (US).

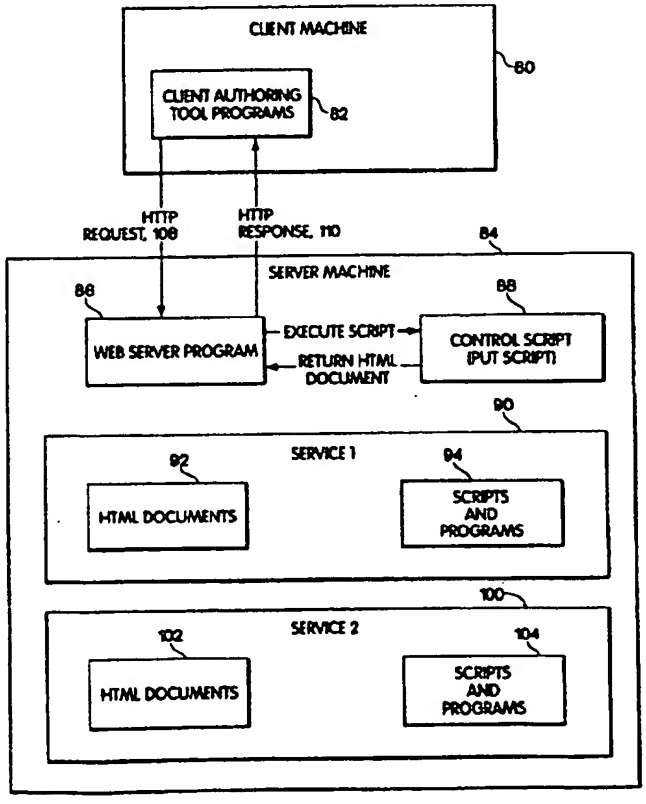
(81) Designated States: JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: COMPUTER SYSTEM AND COMPUTER-IMPLEMENTED PROCESS FOR REMOTE EDITING OF COMPUTER FILES

(57) Abstract

A client/server computer system for remote editing of document objects stored on the server includes a client computer connected to a server computer via a communication channel over which messages are sent in a communication protocol. Typically, the client computer has an operating system with the first file name space and the server computer has an operating system with a second file name space and the first file name space does not include names of files which map to names of files in the second file name space. The connection is preferably a TCP/IP connection providing data transport according to TCP/IP. Messages in the HTTP protocol are preferably used. The client computer sends request messages to the server. A request message may indicate a request for either retrieval or storage of a document object, such as an HTML document or script program. The server receives the request messages and processes them to either store a document object or retrieve a document object and return it to the client in a response message. When the server is an HTTP server, the request messages from the client are processed by a single control script. The messages from the client indicate a desired document object and the action to be performed.



COMPUTER SYSTEM AND COMPUTER-IMPLEMENTED PROCESS FOR REMOTE EDITING OF COMPUTER FILES

Field of the Invention

This invention is related to computer editing systems for editing electronic documents, other information and computer programs. More particularly, this invention is related to computer editing systems for developing on-line services in a client-server information system.

Background of the Invention

An on-line information system typically includes one computer system (the server) that makes information available so that other computer systems (the clients) can access the information. The server manages access to the information, which can be structured as a set of independent on-line services. The server and client communicate via messages conforming to a communication protocol and sent over a communication channel such as a computer network or through a dial-up connection.

Typical uses for on-line services include document viewing, electronic commerce, directory lookup, on-line classified advertisements, reference services, electronic bulletin boards, document retrieval, electronic publishing, technical support for products, and directories of on-line services, among others. The service may make the information available free of charge, or for a fee.

Information sources managed by the server may include files, databases and applications on the server system or on an external system. The information that the server provides simply may be stored on the server, may be converted from other formats manually or automatically, may be computed on the server in response to a client request, may be derived from data and applications on the server or other machines, or may be derived by any combination of these techniques.

The user of an on-line service uses a program running on the client system to access the information managed by the on-line service. Possible user capabilities include viewing, searching, downloading, printing, and filing the information managed by the server. The user may also price, purchase, rent, or reserve services or goods offered through the on-line service.

For example, an on-line service for catalog shopping might work as follows. The user runs a program on the client system and requests a connection to the catalog shopping service using a service name that either is well known or can be found in a directory. The request is

the document, as well as destinations and labels for hypertext links. There are tags for markup elements such as titles, headers, text attributes such as bold and italic, lists, paragraph boundaries, links to other documents or other parts of the same document, in-line graphic images, and many other features.

For example, here are several lines of HTML:

```
Some words are <B>bold</B>, others are <I>italic</I>. Here we start a new
paragraph.<P>Here's a link to
the <A HREF="http://www.vermeer.com">Vermeer Technologies, Inc.</A> home page.
```

This sample document is a hypertext document because it contains a "link" to another document, as provided by the "HREF=." The format of this link will be described below. A hypertext document may also have a link to other parts of the same document. Linked documents may generally be located anywhere on the Internet. When a user is viewing the document using a Web browser (described below), the links are displayed as highlighted words or phrases. For example, using a Web browser, the sample document above would be displayed on the user's screen as follows:

Some words are **bold**, others are *italic*. Here we start a new paragraph.
Here's a link to **Vermeer Technologies, Inc.** home page.

In the Web browser, a link may be selected, for example by clicking on the highlighted area with a mouse. Selecting a link will cause the associated document to be displayed. Thus, clicking on the highlighted text "Vermeer Technologies, Inc." would display that home page.

Another kind of document object on the WWW is a script. A script is an executable program, or a set of commands stored in a file, that can be run by a Web server (described below) to produce an HTML document that is then returned to the Web browser. Typical script actions include library routines or other applications to get information from a file or a database, or initiating a request to get information from another machine, or retrieving a document corresponding to a selected hypertext link. A script is run on the Web server when, for example, the end user selects a particular hypertext link in the Web browser, or submits an HTML form request. Scripts are usually written by a service developer in an interpreted language such as

running the Microsoft MS-DOS operating system and the Microsoft Windows operating environment. The Web server also has a standard interface for running external programs, called the Common Gateway Interface (CGI). A gateway is a program that handles incoming information requests and returns the appropriate document or generates a document dynamically. For example, a gateway might receive queries, look up the answer in an SQL database, and translate the response into a page of HTML so that the server can send the result to the client. A gateway program may be written in a language such as "C" or in a scripting language such as Practical Extraction and Report Language (Perl) or Tcl or one of the Unix operating system shell languages. Perl is described in more detail in Programming Perl, by Larry Wall and Randal L. Schwartz, O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1992. The CGI standard specifies how the script or application receives input and parameters, and specifies how any output should be formatted and returned to the server.

Generally speaking, for security reasons, a Web server machine may limit access to files. For all access to files on the Web server, the Web server program running on the server machine may provide an extra layer of security above and beyond the normal file system and login security procedures of the operating system on the server machine. The Web server program may add further security rules such as: 1) optionally requiring user name and password, completely independent of the normal user name and passwords that the operating system may have on user accounts, 2) allowing definitions of groups of users for security purposes, independent of any user group definitions of the operating system, 3) access control for each document object such that only specified users (with optional passwords) or groups of users are allowed access to the object, or that access is only allowed for clients at specific network addresses, or some combination of these rules, 4) allowing access to the document objects only through a specified subset of the possible HTTP methods, 5) allowing some document objects to be marked as HTML documents, others to be marked as executable scripts that will generate HTML documents, and others to be marked as other types of objects such as images. Access to the online service document objects via a network file system would not conform to the security features of the Web server program and would provide a way to access documents outside of the security provided by the Web server. The Web server program also typically maps document object names that are known to the client to file names on the server file system. This mapping may be arbitrarily complex, and any author or program that tried to access documents on the Web server directly would need to understand this name mapping.

returned to the Web server, which then sends it to the Web browser in an HTTP response message.

Request messages in HTTP contain a "method name" indicating the type of action to be performed by the server, a URL indicating a target object (either document or script) on the Web server, and other control information. Response messages contain a status line, server information, and possible data content. The Multipurpose Internet Mail Extensions (MIME) are a standardized way for describing the content of messages that are passed over a network. HTTP request and response messages use MIME header lines to indicate the format of the message. MIME is described in more detail in MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies, Internet RFC 1341, June 1992.

The request methods defined in the current version of the HTTP protocol include GET, HEAD, POST, PUT, DELETE, LINK, and UNLINK. The GET method requests that the server retrieve the object indicated by the given URL and send it back to the client. If the URL refers to a document, then the server responds by sending back the document. If the URL refers to an executable script, then the server executes the script and returns the data produced by the execution of the script. Web browser programs normally use the GET method to send request messages to the Web server to retrieve HTML documents, which the Web browser then displays on the screen at the client computer.

According to the HTTP specification, the PUT method specifies that the object contained in the request should be stored on the server at the location indicated by the given URL. However, the current server implementations do not follow this specification; they simply handle all PUT requests through a single PUT script, which is generally undefined, and must be created by a service author. Web browsers generally do not use the PUT method.

The POST method sends data, usually the user input parameters from an HTML form, to the server. The POST request also contains the URL of a script to be run on the server. The server runs the script, passing the parameters given in the request, and the script generates HTML output to be returned in the response to the client. In order for a client program to send arbitrary data to the Web server using the current HTTP protocol, the client program must use either the PUT method or the POST method, as these are the only two methods that allow such data transfer to the Web server.

command (step 24). For example, the user may request to view a new document by selecting a hypertext link to a document, by requesting a document from a list of previously visited documents, or by typing in the URL of a document that was obtained by the user through some other means. The browser tests the user command to determine if the user is requesting a new document in step 26. If so, processing continues at step 14 which has already been discussed. If the user is not requesting a new document then the browser tests the command in step 30 to determine if it is a request to exit the program. If so, processing stops. Otherwise the command is a local command that is handled by the browser without sending an HTTP request in step 28. The end user may use local viewing commands such as commands to scroll around in the document, or commands to search for a particular text string in the document. After the browser handles the local command, the browser again waits for the next user command as already discussed, in step 24.

Figure 2 shows the operation of an on-line service on the World Wide Web as seen by the Web server program. When the server is started, it runs continuously, waiting to receive a command over the network connection from a client Web browser program in step 40. The server tests the received command in step 44 to determine if it is a GET request. If it is a GET request, then the server examines the URL contained in the request in step 52 to determine if the URL indicates a static HTML document stored on the server. If the URL does refer to a static document then that document is returned to the Web browser via an HTTP response in step 58. Otherwise the URL indicates a script stored on the server, and the Web server runs the script to produce an HTML document in 56 which is then returned to the Web browser as described before in step 58. If the test of step 44 determines that the command is not a GET request, then the server tests the command in step 48 to determine if it is a POST request. If so, the server retrieves the parameters from the POST request in step 54, which include the URL for the script and the parameters for the script. The server then runs the indicated script in step 56 to generate an HTML document which is then returned to the Web browser as described before in step 58. After an HTML document is returned to the Web browser, processing continues at step 40. If the test of step 48 determines that the command is not a POST request then the server returns an error message to the Web browser in step 50, formatted as an HTML document. The processing continues at step 40 and the server again waits for the next request and the process repeats.

On-line services such as those described above are in high demand. Unfortunately, the task of developing an on-line service is currently one that almost always requires extensive

The second approach has the problem that the server machine and client machine must both run additional programs to allow terminal emulation and remote execution of programs over a network. This adds to the complexity on both machines, and also requires that the service author be familiar with a terminal emulation program which typically has a difficult user interface that is not meant for nonexperts. This approach also adds another route from other machines to the server machine, which may be undesirable for security reasons. As with the first approach, the service author may not have access to the server machine for security reasons, or may not have authorization to write files to the machine.

In the third approach, the service author first transfers existing service documents and scripts from the server machine to a client machine either manually or via a network file transfer program. The author then runs a text or HTML editor program on a client machine to create or modify documents on that machine, and then transfers the completed documents back to the server machine either manually or via a network file transfer program, such as the file transfer protocol (ftp) or kermit, a file transfer method used with terminal emulation programs for communication over a modem.

The third approach is cumbersome because of the need for the separate steps of transferring the documents from the server back to the client, and transferring the documents back to the server after the editing is complete. This approach also has the security problems mentioned above for the other approaches.

Each of these three approaches also has the problem that the file names used for documents by a Web server are not always the same as the actual file names of the documents. An author of an on-line service will need to learn the mappings of file names to the URLs used by the Web server.

There is also the World Wide Web computer program, for use with a NeXT computer, that consists of a client browser program that is able to retrieve files from a Web server, and a client HTML editor that can edit the retrieved file. However, this program is not able to save the edited files to the Web server. Instead, this approach is similar to the third approach discussed above in that a file transfer program is still needed to place the edited document back on the Web server. This approach also is not a complete solution for authoring an on-line service for the Web because the types of documents edited in this manner are limited to static HTML documents which are not processed in any way by the server.

using a client computer, wherein the server computer and the client computer are connected by a communication channel and send messages using a communication protocol. In this process, the client computer sends a request message using the communication protocol over the communication channel to the server requesting a copy of the electronic document. Next, the client receives a response message in the communication protocol from the server and over the communication channel, wherein the response message contains the copy of the electronic document. The client then permits a user to edit the copy of the electronic document at the client computer. To store the electronic document at the server, the client sends a message in the communication protocol including the edited electronic document over the communication channel and to the server, wherein the message includes an indication of a request that the electronic document be stored on the server computer at a particular location. Typically, the server sends a status response message indicating the outcome of the attempt to store the document.

In one embodiment, to overcome these difficulties, the authoring tool uses either PUT or POST HTTP request messages to request retrieval of scripts or documents and to request storage of edited or new documents or scripts. Currently available authoring tools may be readily modified to replace existing "open" and "save" functions to provide this capability. The server may be any standard Web server program such as the CERN server or the NCSA server. Our invention provides a control script which is executed by the server for each incoming PUT or POST HTTP request messages to determine whether a document is to be retrieved or replaced. The authoring tool program uses the HTTP protocol to communicate with a server program running on the server machine. The communication takes the form of the authoring tool sending a request to the server program, the server program executing a control script to perform the indicated action and write the results to an output file, and the server sending a response message back to the authoring client with the output. The output may be either a status message indicating that the action was performed successfully, or contents of a document or script that was retrieved, or an error message indicating why the action could not be performed. The server program may still communicate with browsers at the same time the authoring tool is being used. The server continually listens to incoming messages. If an incoming PUT or POST HTTP request is not from the authoring tool, it is handled in the same manner as other PUT or POST request messages from other client programs such as Web browsers. Otherwise, the server passes the request parameters to the control script. The control script checks the parameters to

Another aspect of the invention is a computer system for enabling a client computer to store a document object on a server computer. The server computer includes a mechanism which receives an HTTP request message from the client computer over a TCP/IP connection, wherein the HTTP request message has content including a copy of the document object and an indication of a location on the server computer to store the copy of the document object and an indication that the client computer requests storage of the document object. The server computer also includes a mechanism which executes a process using the content of the HTTP request message to store the copy of the document object on the server computer according to the indication of the location included in the HTTP request message.

Another aspect of the invention is a server computer in a computer system for enabling a client computer to store a document object on the server computer. The server computer includes a computer-readable and writable storage medium and a TCP/IP mechanism, having an input for receiving a request from the client computer to establish a TCP/IP connection with the client computer and which establishes the TCP/IP connection. The server computer also includes an HTTP server having an input for receiving an HTTP request message from the client computer over the TCP/IP connection via the TCP/IP mechanism, wherein the HTTP request message has content including a copy of the document object and an indication of name used by the HTTP server to retrieve the document object as stored on the storage medium and an indication that the client computer requests storage of the document object, and an output for storing the copy of the document object from the HTTP request message on the computer-readable and writable storage medium as a file according to the name included in the HTTP request message.

Another aspect of the invention is a computer-implemented process for enabling a client computer to store a document object on a server computer. In this process, the client computer establishes a TCP/IP connection with the server computer. Then, the client computer sends an HTTP request message to the server computer over the TCP/IP connection, wherein the HTTP request message has content including a copy of the document object and an indication of a location on the server computer to store the copy of the document object and an indication that the client computer requests storage of the document object. The client computer then receives an HTTP response message from the server computer indicating a result of an attempt by the server computer to store the copy of the document object.

command to the server computer to store the document object as a file on the computer-readable and writable storage medium using the file name in the file name space of the server computer. The server program as executed also has a second output connected to the client computer to provide a response message to the client computer acknowledging storage of the document object.

In the aspects of the invention where the server receives a universal name, the universal name may be a URL or equivalent indicator. Such a universal name is different from a file name used by, for example, FTP because in FTP a message contains actual file name on server and maintains current working directory information of the user who has logged into the server. Additionally, such a universal name is different from a file name such as used by a network file system (NFS). In NFS, a message contains physical file name and may maintain current working directory information. In NFS, the mapping of file names and directories is client-dependent and not client independent. In contrast, using HTTP or other server using universal, or client-independent names, a message uses a URL which contains a protocol://host/path which is passed through a mapping step. Thus, a file is accessible on the server only if it is listed in a mapping table of the server which matches trees of directories to URLs. The URL by itself typically cannot be used without a server to identify the file, unlike file names used in FTP. HTTP also has further rules which allow multiple mappings of subdirectories, unlike file names in NFS. Different security may also be provided on different subdirectories using HTTP. In other words, using HTTP and URLs, a names using a directory x and names using subdirectory x/y may be mapped to different places on server. Additionally, permissions associated with a directory x and subdirectory x/y may be different on the server.

Another aspect of the invention is a computer-implemented process for enabling a client computer to store a document object on a server computer. In this process, a server computer receives a request message from the client computer, wherein the request message has content including a copy of the document object, an indication of an application sending the message, an indication of a location on the server computer, and an indication that the document object is to be stored in the location on the server computer. The server then determines whether the request message is from an authorized source by comparing the indication of the application in the content of the request message with accepted applications. When the request message is from an authorized source, the server stores the document object on the server computer using the location indicated in the request message and sending a response message to the client computer

message is from an authorized source and, when the request message is not from an authorized source, which provides a response message to the client computer not acknowledging storage of the document object.

Another aspect of the invention is a computer-implemented process for enabling a client computer to edit a document object stored on a server computer. In the process, a first HTTP request message is transmitted from the client computer over a TCP/IP connection to the server computer, wherein the first HTTP request message specifies the document object and an indication that the client computer requests retrieval of the document object from the server computer. A process is executed on the server computer using the first HTTP request message which retrieves a copy of the document object. The copy of the document object is transmitted from the server computer to the client computer over a TCP/IP connection in a first HTTP response message. The document object is then edited on the client computer. A second HTTP request message is transmitted from the client computer over a TCP/IP connection to the server computer, wherein the second HTTP request message contains a copy of the document object and an indication of a location on the server computer to store the copy of the document object and an indication that the client computer requests storage of the document object. A process is executed on the server computer using the second HTTP request message which stores the copy of the document object on the server computer according to the indication of the location included in the second HTTP request message.

Another aspect of the invention is a computer system for editing of document objects, comprising a server computer and a client computer. The client computer includes a mechanism for sending a first HTTP request message over a TCP/IP connection to the server computer, wherein the first HTTP request message specifies the document object and an indication that the client computer requests retrieval of the document object from the server computer. The client computer also includes a mechanism for receiving an HTTP response message from the server including the copy of the requested document object and a mechanism for editing the document object on the client computer. The client computer also includes a mechanism for sending a second HTTP request message over a TCP/IP connection to the server computer, wherein the second HTTP request message contains a copy of the edited document object and an indication of a location on the server computer to store the copy of the document object and an indication that the client computer requests storage of the document object. The server computer includes a mechanism for executing a process on the server computer using the first HTTP request

the document object, and then stores the copy of the document object in memory. The second mechanism is for editing the document object stored in the memory. The third mechanism is for storing the edited document object on the server computer. The third mechanism sends a write request message to the server including the copy of the edited document object from the buffer and an indication of a universal name to be used by other clients to access the document object on the server to store the edited document object, then receives an acknowledgment message from the server computer and for communicating the acknowledgment message to a user, and finally allows a user to activate any of the means for retrieving, editing or storing with an integrated user interface.

Another aspect of the invention is a computer-implemented process for enabling a client computer to edit a document object on a server computer. This process involves transmitting a read request message from the client computer to the server computer, wherein the request message includes an indication of the document object, an indication of an application sending the message, and an indication that the document object is to be sent to the client computer. It is then determined, on the server computer, whether the read request message is from an authorized source by comparing the indication of the application in the content of the request message with accepted applications. When the read request message is from an authorized source, a copy of the document object is retrieved on the server computer and a first response message is sent to the client computer including the copy of the document object. When the read request message is not from an authorized source, a response message is sent to the client computer without sending a copy of the document object. The copy of the document object is then edited on the client computer. A write request message is transmitted from the client computer to the server computer, wherein the write request message includes a copy of the document object edited by the client computer, an indication of a location on the server computer, an indication of an application sending the message, and an indication that the document object is to be stored in the location on the server computer. It is then determined, on the server computer, whether the write request message is from an authorized source by comparing the indication of the application in the content of the request message with accepted applications. When the write request message is from an authorized source, the document object is stored on the server computer as a file using the location indicated in the write request message and sending an acceptance response message acknowledging storage of the document object. When the write request message is not from an

server machine may be limited to service authors with a validated user name and password. Thus, by using the HTTP protocol that is already used for on-line services on the World Wide Web. a minimal level of security is provided during the authoring process.

Another advantage of this system is that the authoring process can be used for remote retrieval. editing. and storing of at least two of the types of document objects that comprise an on-line service on the WWW: static HTML documents and script programs that generate HTML documents.

Remote editing of on-line service document objects using the method of this invention also has several advantages over the prior art approaches. such as remote editing over a network file system where a client machine can read and write files on a remote server machine. One advantage is that our invention allows access to the online service document objects only through the Web server program. and thus conforms to the additional security rules implemented by the server program. A second advantage is that since our invention uses the existing HTTP protocol mechanism of the Web server machine. the server machine does not have to run additional software or server programs that are required to implement a network file system or shared access to a remote file system from a client machine. This is an advantage because the additional software would add complexity and would add further possibilities for security loopholes. A third advantage is that our invention allows access to the online service document objects only through the Web server program. and thus conforms to the document object name mapping conventions between URLs and actual file names of the Web server program. The advantage of the method of our invention is that neither client programs nor service authors need to understand this file name mapping and only need to use the URLs.

Brief Description of the Drawing

In the drawing,

Figure 1 shows the prior art sequence of activities on the Web browser during operation of an on-line service on the Web:

Figure 2 shows the prior art sequence of activities on the Web server during operation of an on-line service on the Web:

Figure 3 shows an overview of the authoring framework for remote authoring of on-line services:

Ontario, Canada, and other document editors and program editors, such as VisualBasic, may be used to create documents and scripts and the invention is not limited thereby. For example, the client machine **80** may have dial-up connection to an Internet service provider, typically using a 14.400 baud or faster modem, and using the Trumpet 2.0b application for Microsoft Windows that provides the TCP/IP protocol software running over a SLIP connection using a normal telephone line. In this arrangement, the client machine **80** is connected to the Internet and has its own Internet address.

In this embodiment, the server machine **84** is a Gateway 2000 personal computer with an Intel Pentium processor with a clock speed of 60 MHz, running the BSDi Unix operating system. The Web server program **86** is the CERN Hypertext Transfer Protocol Daemon (HTTPD) server, configured for the Unix operating system. The server machine **84** also has a dial-up connection to an Internet service provider, using a 14.400 baud or faster modem, and using the TCP/IP and SLIP software that comes with the BSDi Unix operating system. Generally speaking, the Web server program is the only program providing access to documents, other than the operating system. It may define groups of users, user names, passwords and file names separately from the operating system of the server machine **84**. With this configuration the client and server machines can establish a TCP/IP connection and exchange messages over the Internet. It should be understood that this embodiment is merely exemplary. A large variety of computers and operating systems have suitable server and client software for communicating using HTTP over the Internet. The client and server machines may also be connected by a local area network (LAN), wide area network (WAN) or may even be the same machine, but with different processes communicating together over a common communication channel. Although communication is generally provided over a TCP/IP connection, other network communication protocols, including other data transport protocols and message protocols may be used. A variety of message protocols for communicating over TCP/IP connections may be used, such as HTTP, FTP, telnet, etc. However, generally speaking, the server and the client do not share files through the file name spaces of their respective operating systems. That is, the file name space of the client does not include or map to names of files on the server. In other words, no pair of file names in the two file names spaces correspond to the same file. More details about setting up client and server machines connected to the Internet and the World Wide Web are discussed in "Setting up Shop on the Internet," by Jeff Frentzen et al., and related articles in Windows

document on the client machine, and save the document back to the Web server. Figure 4 does not attempt to show editing, error handling, and graphical user interface features that are well known in the prior art for text editors and word processing programs. For instance it does not show that the user may be editing several documents independently within several windows, or that the user may abort an editing session at any time without saving a document, or that the user may retrieve, edit, save in one window and then repeat those steps for a different document in the same window. Furthermore, Figure 4 does not attempt to show the editing commands that the user uses to make changes to the document on the client machine because these are well known in the art.

As shown in Figure 4, the service author first runs the authoring tool program on the client machine in step **120**, which provides a graphical user interface for remote editing. The author then asks in step **121** to edit a document or script from a particular on-line service, and identifies the object by specifying a document or script name, the service name, and the address of the Web server where the service is stored, such as by using its URL.

The authoring tool program then sends in step **122** an HTTP PUT request to the Web server where the service is stored. The structure of this PUT request is shown in Figure 6a which is described in more detail below, and includes header fields for authorization, the MIME version number, the request content type, and the content length in bytes. The body of the request includes a method command field that identifies the request as a retrieve request and a URL command field that gives the URL for the document or script object to be retrieved.

When the Web server program receives an HTTP PUT request in step **124**, it passes it to the control script **88**. The control script examines the parameters of the retrieve request, and writes an output file in step **126** that contains either the requested document or script, or an error message for the service author indicating why the request could not be satisfied. Generally speaking, the Web server program translates the URL into a file name in the file name space used by the operating system of the server machine. Such mappings are usually found in a configuration file for the server. The control script can also be made to perform such translation and maintain its own configuration file of mappings. This can be done so that the server does not have to be reinitialized when mappings change, for example, by the creation of a new service. The detailed operation of the control script is shown in Figure 5, and is explained further below. When the control script has finished execution, the Web server sends the resulting output file to the client authoring tool via an HTTP response in step **128**.

replaced by the number of characters in the content part of the message (the part of the message that follows the Content-length line). The HTTP requests include a single request line, followed by header fields in the HTTP header, followed by a blank line, followed by command fields in the HTTP request body, followed by optional data in the HTTP request body. Header fields have a header field name followed by colon (":") and a header field value. Command fields have a command field name followed by a colon (":") and a command field value. The format of the request line and header fields is defined by the HTTP protocol specification, while the format of the HTTP request body, including the command fields, is defined by the method of our invention.

Figure 6a shows the HTTP request message format for a "retrieve" request. The first line **201** is the request line, comprising the "PUT" method name **211**, followed by the URL for the object to be retrieved **221**, followed by the HTTP protocol version number **231**. The second line **202** is the authorization header field, comprising the header field name **212**, e.g., "Authorization," followed by the header field value **222**, in this case "Basic dmVybWVlcjpEb250Rm9yZ2V0VGhpcw==." This authorization field may be used to carry passwords for protection purposes by the Web server. For example, the Web server may only allow use of the PUT request message by particular users. Also, the Web server could look to the password when a user attempts to write to a file using the Web server. These and other security devices may be used in connection with this invention. The third line **203** is the MIME version header field, comprising the header field name **213**, e.g., "MIME_version," followed by the header field value **223**, e.g., "1.0". The fourth line **204** is the content type header field, comprising the header field name **214**, e.g., "Content-type," followed by the header field value **224**, e.g., "application/x-vermeer." The fifth line **205** is the content length header field, comprising the header field name **215**, e.g., "Content-length," followed by the header field value **225** giving the length in bytes of the HTTP request body. The sixth line **206** is the blank line that separates the header fields from the HTTP request body. The seventh line **207** is the method command field, comprising the command field name **217**, e.g., "method," followed by the command field value **227**, "retrieve." The eighth line **208** is the URL command field, comprising the command field name **218**, "url," followed by the command field value **228** giving the URL for the object on the server machine that is to be retrieved.

Figure 6b shows the HTTP request message format for a "replace" request. The first line **301** is the request line, comprising the "PUT" method name **311**, followed by the URL **321**

correct. If the command fields are not correct, then the control script writes an error message to the output file in step **170** and the control script terminates. If the command fields are acceptable, then the test in step **168** yields the answer "Yes" and the control script performs a test to determine if the "method" command field value indicates a retrieve request (step **172**). If so, then another test is performed to determine if the file given in the "URL" command field value exists and is readable (step **174**). If the result is "Yes" then the control script sets the content type by examining the file name extension in step **176**. The control script then writes the HTTP response header containing the content type, and the data content of the indicated file, to the output file in step **178**, and the control script terminates. If the test in step **174** yields the answer "No" then the control script writes an error message to the output file in step **180** and the control script terminates.

If the "method" command field does not indicate a "retrieve request then the test in step **172** yields the answer "No" and a test is performed to see if the command field indicates a "replace" request (step **182**). If the "method" command field does not indicate a "replace" request then the test **182** yields the answer "No," the control script writes an error message to the output file in step **194**, and the control script terminates. Otherwise, if the "method" command field indicates a "replace" request, then another test is performed in step **184** to determine if the file given in the "URL" command field value may be written by the authoring tool and this user. For example, the file may be written if the file does not currently exist, or if the file exists and the service author has authorization to write the file. Authorization may be determined by the Web server, to see if the user attempting to write to a file has provided a correct password. Authorization may also be determined by the underlying operating system, to see if the Web server has authorization to write to the file specified by the URL. If the result of the test in step **184** is "Yes" then the control script stores the data of the HTTP request body in the file indicated by the "URL" command field value (step **186**). The control script then sets the appropriate file permissions on the new file in step **188**, writes the status message indicating the result of the command into the output file in step **190**, and the control script terminates. If the file cannot be written then the test **184** yields the answer "No," the control script writes an error message to the output file in step **192**, and the control script terminates.

An advantage of this system is that the client authoring tool does not need to map the file name space of the server to its own file name space. This arrangement is particularly advantageous in large networks, such as the Internet, where there may be many authors of many

APPENDIX

```
# !/usr/local/bin/tclsh

#
# file put_script
#
# This is a put_script for a CERN server that supports the following
# operations in the Vermeer authoring tool client:
#       retrieve - retrieve a document
#       replace         - replace a previously existing document
#                         (error if it didn't previously exist
#                         or if date/time is later than that passed in)
#       create          - create a new document
#                         (error if it previously existed)
#       parse           - parse a basic document and return tcl
#       index           - index a document
#                         (indexes should still be recreated regularly).
#       listservices    - return a list of services on the server.
#       linkmap         - return a linkmap of documents on the server.
#
# This script is triggered by an HTTP PUT method. but HTTP POST method
# could also be used.


#
# proc DBG 'debug-string'
#
#if debugging is enabled (/tmp/debug_put_script exists) then
#write the debug-string to /tmp/put_script.dbg
#
proc DBG str {
   global dbgon
```

```
</body>
</html>"

    exit 0
}

#
# proc mktempfile 'File' 'Filename'
#
# Create and open for writing a temporary file in $TMPDIR
#
proc mktempfile { File Filename } {
    upvar 1 $File file
    upvar 1 $Filename filename
    global tmpdir
    global tmpcount
    global env

    if![info exists tmpdir] {
        if [info exists env(TMPDIR)] {
            set tmpdir$env(TMPDIR)
        } else {
            set tmpdir /tmp
        }
            set tmpcount 0
    }

    # Need a way to ensure exclusive create...
    # (and maybe put a process id in here...)
    set filename $tmpdir/VTtmp$tmpcount
    incr tmpcount
    while {[file exists $filename] == 1} {
```

```
    if {$contentlength <=0} {
        return -1
    }

    if {[gets stdin line] <0} {
        return -1
    }

    # dont't fully understand CRLF issues. but for now...
    incr contentlength [expr 0- ([string length $line] + 1)]
    set line [string trimright $line "\r"]

    DBG "input - $line"

    return 0
}

#
# Execution starts here
#

if {[catch {

DBG "in put_script"

#
# Make sure we have a document of type "application/x-vermeer"
#
if ![info exists env(CONTENT_TYPE)] {
    error "no content-type\nneed content-type: application/x-vermeer"
}
```

```
    set ix [string last $env(PATH_INFO) $env(PATH_TRANSLATED)]
    set pathroot [string range $env(PATH_TRANSLATED) 0 [expr $ix-1] ]
  } else {
    error "cannot determine root of document tree "
  }

  set molisaroot [format "%s/Molisa" $pathroot]

  #
  # Most methods require a target url.
  # Get and check it if it exists.
  #

  if [info exists headers(url)] {
    set url $headers(url)
    set fullurl $pathroot$url
    set urlextension [file extension $url]

    if { 0 != [string first "/Molisa" $url] } {
        error "document is not underneath /Molisa - $url"
    }
  } else {
    set url ""
    set fullurl ""
    set urlextension ""
  }

  #
  # Get the method
  #
  if ![info exists headers(method)] {
    error "no method header"
```

```
    puts stdout "ENDOFLIST "
    exit 0
  } elseif {0 == [string compare $method index] } {
  #
  # Index
  #
    DBG "entered index "

    set indexer /usr/local/bin/waisindex
    if ![file exits $fullurl] {
        error "document does not exist - $url"
    }

    if ![file exists $indexer] {
        error "indexer does not exist - $indexer"
    }
    if ![file readable $fullurl] {
        error "document not readable - $url"
    }

    if [file isdirectory $fullurl] {
        DBG "directory index branch"
        set service [file tail $url]
  #  We use a Web directory for the html pages. and a parallel wais
  #  directory, rather than mixing wais index files into the Web hierarchy
        set www [file dirname $pathroot]/wais/$service

        catch { exec $indexer - mem 2 -1 l -export -d $www \
          -t URL $molisaroot/$service http://zorch.tiac.net/Molisa/$service \
          -r $molisaroot/$service} result
        DBG "result = -$result-"
      } else {
```

```
# Retrieve
#
   if ![file exists $fullurl] {
        error "document does not exist - $url"
}
if ![file readable $fullurl] {
        error "document not readalbe -$url"
}

if { 0 == [string compare $urlextension .html] } {
        set contenttype text/html
} elseif { 0 == [string compare $urlextension .bas] } {
        set contenttype text/x-basic
} elseif { 0 == [string compare $urlextension .tcl] } {
        set contenttype text/x-tcl
} else {
        set contenttype application/binary
   }

   puts stdout "Content-Type: $contenttype\n"

   set input [open $fullurl r]
   while {[gets $input line] >=0} {
        puts stdout $line
   }
   clost $input

   exit 0
} elseif {(0 == [string compare $method replace]) ||
        (0 == [string compare $method create]) } {
#
# Replace or create
```

```
        exec >@ $output $molisaroot/internal/read $contentlength
    }
    close $output

    # Quick detection of executable files
    # If the directory name is "Scripts" make it executable
    if {0 == [string compare Scripts [file tail [file dirname $fullurl]]]} {
        exec chmod a+x $fullurl
    }

    puts stdout "Content-Type: text/html

<html>
<head>
<title>Create/replace succeeded</title>
</head>
<body>
<h1>Create/replace succeeded </h1>
</body>
</html>"

    exit 0
} elseif {0 == [string compare $method parse] } {
#
# Parse
#
    mktempfile basicFile baseFileName

    while { [readline line] >=0 } {
        puts $basicFile $line
    }
    close $basicFile
```

```
            set tclFile [open $tclFileName "r"]
            while {[gets $tclFile line] >=0} {
               puts stdout $line
            }
            puts stdout "</pre>
</body>
</html>"
   }

   exec rm $basicFileName $tclFileName
}

} vfr_put_result] } {
   DBG "put caught - '$vfr_put_result'"
   puts stdout "Content-Type: text/html

<HEAD><TITLE>Put Error</TITLE></HEAD><BODY>
<H1>Put Error</H1><pre>$vfr_put_result</pre></body>"
   exit 0
}
```

response message to the client program, indicating one of acknowledgment that the document was successfully saved and giving an error indication.

5. The process of claim 1, wherein the document object is part of an online service on the World Wide Web.

6. The process of claim 1, wherein the document object is comprised of an HTML file.

7. The process of claim 1, wherein the document object is a computer program written in a computer programming language.

8. The process of claim 1, wherein the script is a single computer program which processes both retrieve and store requests and is called by the server in response to either a store or a retrieve request from the client.

9. The process of claim 1, wherein the HTTP protocol includes a named message method which indicates transfer of arbitrary data from the client to the server and wherein the client sends the server a message including the name of the named message method for both retrieve and store requests.

10. The process of claim 9, wherein the named message method is an HTTP "PUT" message.

11. The process of claim 9, wherein the named message method is an HTTP "POST" message.

12. The process of claim 1, wherein the server and client are on the same computer.

13. The process of claim 1, wherein the server and the client are on separate computers interconnected by a network.

14. The process of claim 13, wherein the network is a local area network.

are connected via a communication channel using a communication protocol, the process comprising the steps, performed by the client computer, of:

sending a request message in the communication protocol and over the communication channel to the server requesting a copy of the electronic document using a name mappable to the file name space of the server and not mappable to the file name space of the client,

receiving a response message in the communication protocol from the server and over the communication channel, wherein the response message contains the copy of the electronic document,

permitting editing of the copy of the electronic document at the client computer, and

sending a message in the communication protocol including the edited electronic document over the communication channel and to the server, wherein the message includes an indication of a request that the electronic document be stored on the server computer using a name mappable to the file name space of the server and not mappable to the file name space of the client.

22. A client computer system for use in connection with a server computer connected to the client computer via a communication channel using a communication protocol, and wherein the client computer has an operating system with a first file name space and the server computer has an operating system with a second file name space and the first file name space does not include names of files which map to names of files in the second file name space, the client computer system comprising:

means for sending a request message over the communication channel in the communication protocol to the server for a copy of the electronic document,

means for receiving a response message in the communication protocol from the server and over the communication channel, wherein the response message contains the copy of the electronic document,

means for permitting editing of the copy of the electronic document at the client computer, and

means for sending a message in the communication protocol including the edited electronic document over the communication channel and to the server, wherein the message includes an indication of a request that the electronic document be stored on the server computer.

receiving a message in the communication protocol including an edited electronic document over the communication channel and from the client, wherein the message includes an indication of a request that the electronic document be stored on the server computer, and

storing the edited electronic document on the server computer system.

25. A computer-implemented process for remotely editing an electronic document stored on a server, wherein the client and the server communicate over a communication channel using a communication protocol, and wherein the client computer has an operating system with a first file name space and the server computer has an operating system with a second file name space and the first file name space does not include names of files which map to names of files in the second file name space, comprising the steps of:

the client establishing the communication channel with the server,

the client sending a message in the communication protocol over the communication channel to the server, wherein the message specifies the electronic document and an indication that the client requests retrieval of the electronic document,

the server receiving the message and checking authentication,

the server retrieving a copy of the electronic document if access is authenticated,

the server sending the copy of the electronic document to the client over the communication channel in a message in the communication protocol,

the client receiving the message from the server including the copy of the electronic document,

the client permitting editing of the copy of the electronic document,

the client sending a message over the communication channel and in the communication protocol to the server, wherein the message contains a copy of the edited document and an indication of a location on the server to store the copy of the edited document and an indication that the client requests storage of the edited document,

the server receiving the message and storing the copy of the edited document on the server at the location specified in the message, and

the server sending a message acknowledging an attempt at storage of the edited document.

26. A computer system for remotely editing an electronic document, comprising:

determining whether the request message is for retrieval of a document object when the request message is from the authoring tool,

sending a response message to the authoring tool including the document object when the request message is for retrieval,

determining whether the request message is for storage of a document object when the request message is from the authoring tool.

storing the document object when the request message is for storage, and

sending a response message to the authoring tool acknowledging storage of the document object when the request message is for storage.

28. A process for saving a document object on a remote server. and wherein the client computer has an operating system with a first file name space and the server computer has an operating system with a second file name space and the first file name space does not include names of files which map to names of files in the second file name space. comprising the steps of:

sending a request message to the remote server. wherein the request message includes the document object. an indication of a location on the remote server and an indication that the document object is to be stored on the remote server.

the remote server receiving the request message and converting the indication of the location into a file name. and storing the document object as a file using the file name. and

the remote server sending a response message acknowledging storage of the document object.

29. A client computer for use in a client/server computer system for remotely editing document objects stored on the server computer, the client computer comprising:

an editing system having inputs connected to receive editing commands. a memory for storing the document object while it is edited in response to the editing commands and an outputfor displaying the document object to the user during editing,

a retrieve request message processor having an input connected to receive an indication of a document object on the server to be retrieved and an output providing a retrieve request message, including an indication of the document object. to a communication channel connected to the server computer:

32. A computer-implemented process for enabling a client computer to store a document object on a server computer, comprising the steps, performed by the server computer, of:

receiving an HTTP request message from the client computer over a TCP/IP connection, wherein the HTTP request message has content including a copy of the document object and an indication of a location on the server computer to store the copy of the document object and an indication that the client computer requests storage of the document object; and

executing a process using the content of the HTTP request message to store the copy of the document object on the server computer according to the indication of the location included in the HTTP request message.

33. In a computer system for enabling a client computer to store a document object on a server computer, the server computer comprising:

means for receiving an HTTP request message from the client computer over a TCP/IP connection, wherein the HTTP request message has content including a copy of the document object and an indication of a location on the server computer to store the copy of the document object and an indication that the client computer requests storage of the document object; and

means for executing a process using the content of the HTTP request message to store the copy of the document object on the server computer according to the indication of the location included in the HTTP request message.

34. In a computer system for enabling a client computer to store a document object on a server computer, the server computer comprising:

a computer-readable and writable storage medium;

a TCP/IP mechanism, having an input for receiving a request from the client computer to establish a TCP/IP connection with the client computer and which establishes the TCP/IP connection; and

an HTTP server having an input for receiving an HTTP request message from the client computer over the TCP/IP connection via the TCP/IP mechanism, wherein the HTTP request message has content including a copy of the document object and an indication of name used by the HTTP server to retrieve the document object as stored on the storage medium and an indication that the client computer requests storage of the document object, and an output for storing the copy of the document object from the HTTP request message on the computer-

to retrieve the document object, and an indication that the document object is to be stored in the indicated location on the server computer;

means for converting the indicated name into a file name in a file name space of the server computer and storing the document object as a file on the server computer using the file name in the file name space of the server computer; and

means for sending a response message to the client computer acknowledging storage of the document object.

38. In a computer system for enabling a client computer to store a document object on a server computer, the server computer comprising:

a computer-readable and writable storage medium:

a server program executed on the server computer having:

a. an input connected to receive a request message from the client computer, wherein the request message has content including a copy of the document object, an indication of a name to be used by the server program to retrieve the document object which is a universal name used by other clients to retrieve the document object, and an indication that the document object is to be stored on the computer-readable and writable storage medium;

b. a first output for providing a file name in a file name space of the server computer converted from the indicated name in the received message and a command to the server computer to store the document object as a file on the computer-readable and writable storage medium using the file name in the file name space of the server computer: and

c. a second output connected to the client computer to provide a response message to the client computer acknowledging storage of the document object.

39. A computer-implemented process for enabling a client computer to store a document object on a server computer, comprising the steps, performed by the server computer, of:

receiving a request message from the client computer, wherein the request message has content including a copy of the document object, an indication of an application sending the message, an indication of a location on the server computer, and an indication that the document object is to be stored in the location on the server computer:

determining whether the request message is from an authorized source by comparing the indication of the application in the content of the request message with accepted applications:

b. a comparator having a first input for receiving the indication of the application and a second input for receiving an indication of an authorized source and an output providing an indication of whether the request message is from an authorized source;

c. a first output which provides a command to the server computer when the request message is from an authorized source to store the document object on the computer readable and writable storage medium as a file using the name indicated in the request message; and

d. a second output which provides a response message to the client computer acknowledging storage of the document object when the request message is from an authorized source and, when the request message is not from an authorized source, which provides a response message to the client computer not acknowledging storage of the document object.

42. A computer-implemented process for enabling a client computer to edit a document object stored on a server computer, comprising the steps of:

transmitting a first HTTP request message from the client computer over a TCP/IP connection to the server computer, wherein the first HTTP request message specifies the document object and an indication that the client computer requests retrieval of the document object from the server computer;

executing a process on the server computer using the first HTTP request message which retrieves a copy of the document object;

transmitting the copy of the document object from the server computer to the client computer over a TCP/IP connection in a first HTTP response message;

editing the document object on the client computer;

transmitting a second HTTP request message from the client computer over a TCP/IP connection to the server computer, wherein the second HTTP request message contains a copy of the document object and an indication of a location on the server computer to store the copy of the document object and an indication that the client computer requests storage of the document object;

executing a process on the server computer using the second HTTP request message which stores the copy of the document object on the server computer according to the indication of the location included in the second HTTP request message.

43. A computer system for editing of document objects, comprising:

converting, on the server computer, the indication of the document object to a file name in a file name space of the server computer and retrieving a copy of the document object from the server computer using the file name;

sending the copy of the document object from the server computer to the client computer in a read response message;

editing the copy of the document object on the client computer;

sending a write request message from the client computer to the server computer, wherein the write request message includes a copy of the document object, a universal name for the document object to be used by other clients to access the document object, and an indication that the document object is to be stored on the server computer to be accessible by the client computer using the universal name in the message;

converting on the server computer the indicated name into a file name in the file name space of the server computer and storing the document object as a file using the file name in the file name space; and

sending a write response message to the client computer acknowledging storage of the document object.

45. A client application for use on a client computer for editing a document object stored on a remote server computer, comprising, with an integrated user interface providing access to simultaneously to:

a. means for retrieving the document object from the server computer, including

i) means for sending a read request message to the server computer including an indication of the document object using a universal name to be used by other clients to access the document object, and an indication that a copy of the document object is to be sent by the server computer to the client computer.

ii) means for receiving a read response message from the server computer including the copy of the document object, and

iii) storing the copy of the document object in memory,

b. means for editing the document object stored in the memory;

c. means for storing the edited document object on the server computer, including

when the write request message is from an authorized source, storing the document object on the server computer as a file using the location indicated in the write request message and sending an acceptance response message acknowledging storage of the document object; and

when the write request message is not from an authorized source, sending a denial response message to the client computer without storing the document object on the server computer.

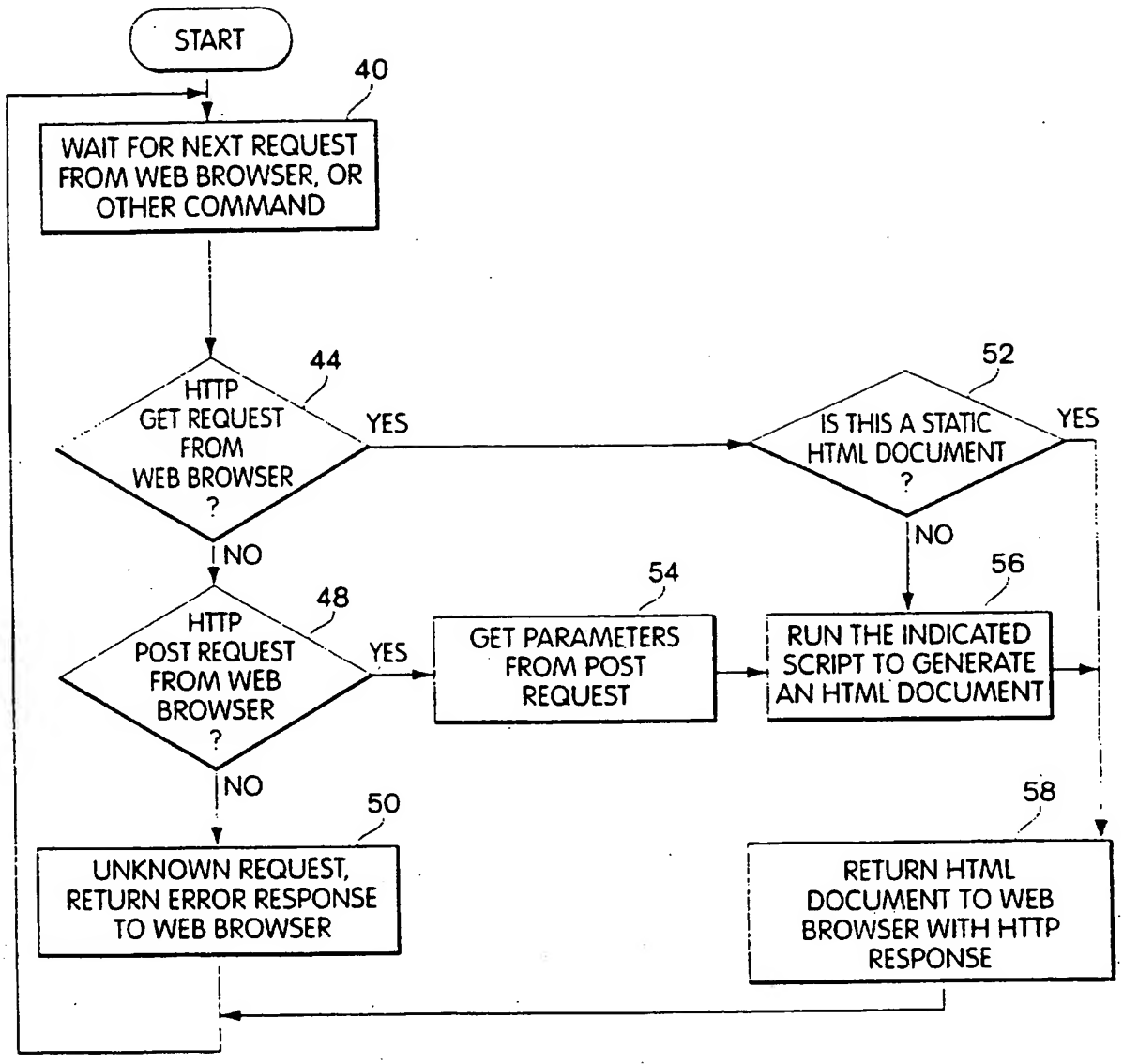


Fig. 2
(Prior Art)

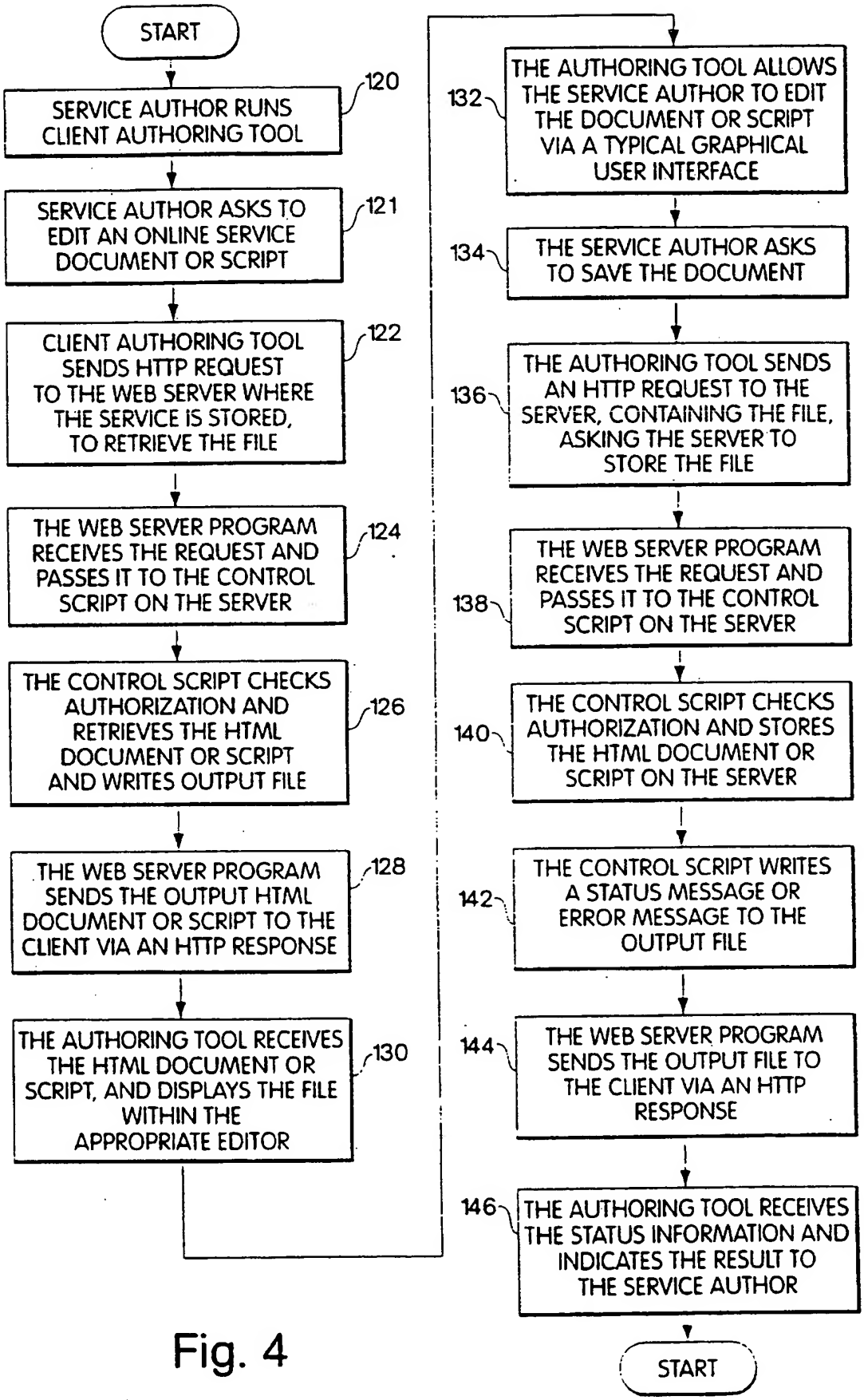

START
SERVICE AUTHOR RUNS CLIENT AUTHORING TOOL
120
SERVICE AUTHOR ASKS TO EDIT AN ONLINE SERVICE DOCUMENT OR SCRIPT
121
CLIENT AUTHORING TOOL SENDS HTTP REQUEST TO THE WEB SERVER WHERE THE SERVICE IS STORED, TO RETRIEVE THE FILE
122
THE WEB SERVER PROGRAM RECEIVES THE REQUEST AND PASSES IT TO THE CONTROL SCRIPT ON THE SERVER
124
THE CONTROL SCRIPT CHECKS AUTHORIZATION AND RETRIEVES THE HTML DOCUMENT OR SCRIPT AND WRITES OUTPUT FILE
126
THE WEB SERVER PROGRAM SENDS THE OUTPUT HTML DOCUMENT OR SCRIPT TO THE CLIENT VIA AN HTTP RESPONSE
128
THE AUTHORING TOOL RECEIVES THE HTML DOCUMENT OR SCRIPT, AND DISPLAYS THE FILE WITHIN THE APPROPRIATE EDITOR
130
132
THE AUTHORING TOOL ALLOWS THE SERVICE AUTHOR TO EDIT THE DOCUMENT OR SCRIPT VIA A TYPICAL GRAPHICAL USER INTERFACE
134
THE SERVICE AUTHOR ASKS TO SAVE THE DOCUMENT
136
THE AUTHORING TOOL SENDS AN HTTP REQUEST TO THE SERVER, CONTAINING THE FILE, ASKING THE SERVER TO STORE THE FILE
138
THE WEB SERVER PROGRAM RECEIVES THE REQUEST AND PASSES IT TO THE CONTROL SCRIPT ON THE SERVER
140
THE CONTROL SCRIPT CHECKS AUTHORIZATION AND STORES THE HTML DOCUMENT OR SCRIPT ON THE SERVER
142
THE CONTROL SCRIPT WRITES A STATUS MESSAGE OR ERROR MESSAGE TO THE OUTPUT FILE
144
THE WEB SERVER PROGRAM SENDS THE OUTPUT FILE TO THE CLIENT VIA AN HTTP RESPONSE
146
THE AUTHORING TOOL RECEIVES THE STATUS INFORMATION AND INDICATES THE RESULT TO THE SERVICE AUTHOR
START

Fig. 4

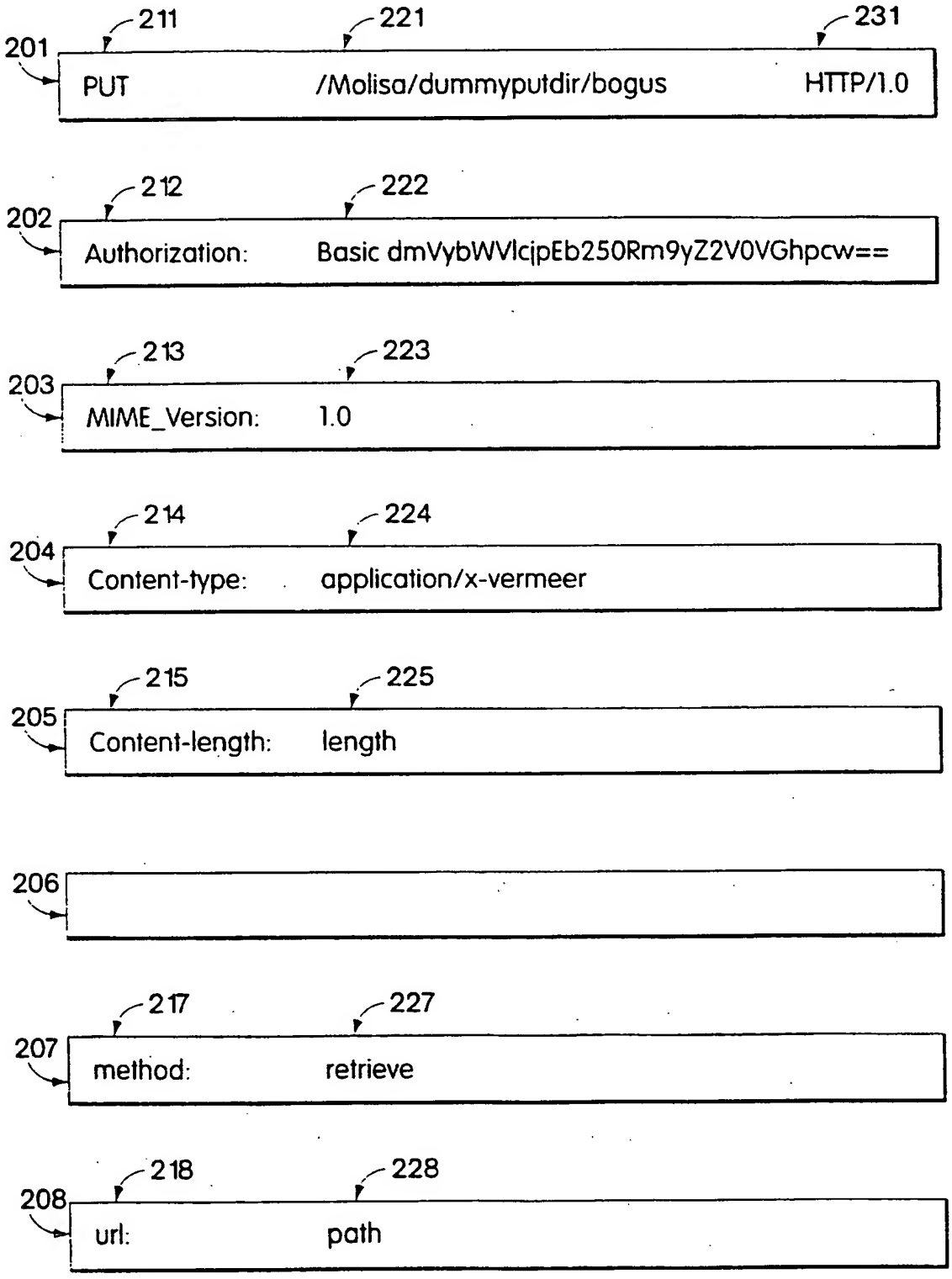

201
211
221
231
PUT
/Molisa/dummyputdir/bogus
HTTP/1.0
202
212
222
Authorization:
Basic dmVybWVlcjpEb250Rm9yZ2V0VGhpcw==
203
213
223
MIME_Version:
1.0
204
214
224
Content-type:
application/x-vermeer
205
215
225
Content-length:
length
206
207
217
227
method:
retrieve
208
218
228
url:
path

Fig. 6A

INTERNATIONAL SEARCH REPORT

Interr nal Application No
PCT/US 96/03650

A. CLASSIFICATION OF SUBJECT MATTER
IPC 6 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 6 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
P,A	THIRD INTERNATIONAL WORLD-WIDE WEB CONFERENCE, DARMSTADT, GERMANY, 10-14 APRIL 1995, vol. 27, no. 6, ISSN 0169-7552, COMPUTER NETWORKS AND ISDN SYSTEMS, APRIL 1995, NETHERLANDS, pages 841-847, XP000565183 PAOLI J: "Cooperative work on the network: edit the WWW!" see the whole document --- -/--	1-46

[X] Further documents are listed in the continuation of box C.

[ ] Patent family members are listed in annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
10 July 1996	25.07.96

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax (+31-70) 340-3016	Katerbau, R